ENTROPY CODING BY ADAPTING CODING BETWEEN LEVEL AND RUN-LENGTH/LEVEL MODES

FIELD

The present invention relates to adaptive entropy encoding of audio data. For example, an audio encoder switches between Huffman coding of direct levels of quantized audio data and arithmetic coding of run lengths and levels of quantized audio data.

RELATED APPLICATION DATA

This application claims the benefit of U.S. Provisional Patent Application No. 60/408,538, filed September 4, 2002, the disclosure of which is hereby incorporated herein by reference.

The following concurrently filed U.S. patent applications relate to the present application: 1) U.S. Provisional Patent Application Serial No. 60/408,517, entitled, "Architecture and Techniques for Audio Encoding and Decoding," filed September 4, 2002, the disclosure of which is hereby incorporated by reference; and 2) U.S. Provisional Patent Application Serial No. 60/408,432, entitled, "Unified Lossy and Lossless Audio Compression," filed September 4, 2002, the disclosure of which is hereby incorporated by reference.

BACKGROUND

With the introduction of compact disks, digital wireless telephone networks, and audio delivery over the Internet, digital audio has become commonplace. Engineers use a variety of techniques to process digital audio efficiently while still maintaining the quality of the digital audio. To understand these techniques, it helps to understand how audio information is represented and processed in a computer.

I. Representation of Audio Information in a Computer

A computer processes audio information as a series of numbers representing the audio information. For example, a single number can represent an audio sample, which is an amplitude value (i.e., loudness) at a particular time. Several factors affect the quality of the audio information, including sample depth, sampling rate, and channel mode.

Sample depth (or precision) indicates the range of numbers used to represent a sample. The more values possible for the sample, the higher the quality because the number can capture more subtle variations in amplitude. For example, an 8-bit sample has 256 possible values, while a 16-bit sample has 65,536 possible values.

The sampling rate (usually measured as the number of samples per second) also affects quality. The higher the sampling rate, the higher the quality because more frequencies of sound can be represented. Some common sampling rates are 8,000, 11,025, 22,050, 32,000, 44,100, 48,000, and 96,000 samples/second.

Table 1 shows several formats of audio with different quality levels, along with corresponding raw bitrate costs.

Quality	Sample Depth (bits/sample)	Sampling Rate (samples/second)	Mode	Raw Bitrate (bits/second)
Internet telephony	8	8,000	mono	64,000
Telephone	8	11,025	mono	88,200
CD audio	16	44,100	stereo	1,411,200
High quality audio	16	48,000	stereo	1,536,000

Table 1: Bitrates for different quality audio information

As Table 1 shows, the cost of high quality audio information such as CD audio is high bitrate. High quality audio information consumes large amounts of computer storage and transmission capacity. Companies and consumers increasingly depend on computers, however, to create, distribute, and play back high quality audio content.

II. Audio Compression and Decompression

Many computers and computer networks lack the resources to process raw digital audio. Compression (also called encoding or coding) decreases the cost of storing and transmitting audio information by converting the information into a lower bitrate form. Compression can be lossless (in which quality does not suffer) or lossy (in which quality suffers but bitrate reduction through lossless compression is more dramatic). Decompression (also called decoding) extracts a reconstructed version of the original information from the compressed form.

Generally, the goal of audio compression is to digitally represent audio signals to provide maximum signal quality with the least possible amount of bits. A conventional audio encoder/decoder ["codec"] system uses subband/transform coding, quantization, rate control, and variable length coding to achieve its compression. The quantization and other lossy compression techniques introduce potentially audible noise into an audio signal. The audibility of the noise depends on how much noise there is and how much of the noise the listener perceives. The first factor relates mainly to objective quality, while the second factor depends on human perception of sound. The conventional audio encoder then losslessly compresses the quantized data using variable length coding to further reduce bitrate.

A. Lossy Compression and Decompression of Audio Data

Conventionally, an audio encoder uses a variety of different lossy compression techniques. These lossy compression techniques typically involve frequency transforms, perceptual modeling/weighting, and quantization. The corresponding decompression involves inverse quantization, inverse weighting, and inverse frequency transforms.

Frequency transform techniques convert data into a form that makes it easier to separate perceptually important information from perceptually unimportant information. The less important information can then be subjected to more lossy compression, while

the more important information is preserved, so as to provide the best perceived quality for a given bitrate. A frequency transformer typically receives the audio samples and converts them into data in the frequency domain, sometimes called frequency coefficients or spectral coefficients.

Most energy in natural sounds such as speech and music is concentrated in the low frequency range. This means that, statistically, higher frequency ranges will have more frequency coefficients that are zero or near zero, reflecting the lack of energy in the higher frequency ranges.

Perceptual modeling involves processing audio data according to a model of the human auditory system to improve the perceived quality of the reconstructed audio signal for a given bitrate. For example, an auditory model typically considers the range of human hearing and critical bands. Using the results of the perceptual modeling, an encoder shapes noise (e.g., quantization noise) in the audio data with the goal of minimizing the audibility of the noise for a given bitrate. While the encoder must at times introduce noise (e.g., quantization noise) to reduce bitrate, the weighting allows the encoder to put more noise in bands where it is less audible, and vice versa.

Quantization maps ranges of input values to single values, introducing irreversible loss of information or quantization noise, but also allowing an encoder to regulate the quality and bitrate of the output. Sometimes, the encoder performs quantization in conjunction with a rate controller that adjusts the quantization to regulate bitrate and/or quality. There are various kinds of quantization, including adaptive and non-adaptive, scalar and vector, uniform and non-uniform. Perceptual weighting can be considered a form of non-uniform quantization.

Inverse quantization and inverse weighting reconstruct the weighted, quantized frequency coefficient data to an approximation of the original frequency coefficient data. The inverse frequency transformer then converts the reconstructed frequency coefficient data into reconstructed time domain audio samples.

B. Lossless Compression and Decompression of Audio Data

Conventionally, an audio encoder uses one or more of a variety of different lossless compression techniques. In general, lossless compression techniques include run-length encoding, Huffman encoding, and arithmetic coding. The corresponding decompression techniques include run-length decoding, Huffman decoding, and arithmetic decoding.

Run-length encoding is a simple, well-known compression technique used for camera video, text, and other types of content. In general, run-length encoding replaces a sequence (i.e., run) of consecutive symbols having the same value with the value and the length of the sequence. In run-length decoding, the sequence of consecutive symbols is reconstructed from the run value and run length. Numerous variations of run-length encoding/decoding have been developed. For additional information about run-length encoding/decoding and some of its variations, see, e.g., Bell et al., Text Compression, Prentice Hall PTR, pages 105-107, 1990; Gibson et al., Digital Compression for Multimedia, Morgan Kaufmann, pages 17-62, 1998; U.S. Patent No. 6,304,928 to Mairs et al.; U.S. Patent No. 5,883,633 to Gill et al; and U.S. Patent No. 6,233,017 to Chaddha.

Run-level encoding is similar to run-length encoding in that runs of consecutive symbols having the same value are replaced with run lengths. The value for the runs is the predominant value (e.g., 0) in the data, and runs are separated by one or more levels having a different value (e.g., a non-zero value).

The results of run-length encoding (e.g., the run values and run lengths) or run-level encoding can be Huffman encoded to further reduce bitrate. If so, the Huffman encoded data is Huffman decoded before run-length decoding.

Huffman encoding is another well-known compression technique used for camera video, text, and other types of content. In general, a Huffman code table associates variable-length Huffman codes with unique symbol values (or unique combinations of values). Shorter codes are assigned to more probable symbol values,

and longer codes are assigned to less probable symbol values. The probabilities are computed for typical examples of some kind of content. Or, the probabilities are computed for data just encoded or data to be encoded, in which case the Huffman codes adapt to changing probabilities for the unique symbol values. Compared to static Huffman coding, adaptive Huffman coding usually reduces the bitrate of compressed data by incorporating more accurate probabilities for the data, but extra information specifying the Huffman codes may also need to be transmitted.

To encode symbols, the Huffman encoder replaces symbol values with the variable-length Huffman codes associated with the symbol values in the Huffman code table. To decode, the Huffman decoder replaces the Huffman codes with the symbol values associated with the Huffman codes.

In scalar Huffman coding, a Huffman code table associates a single Huffman code with one value, for example, a direct level of a quantized data value. In vector Huffman coding, a Huffman code table associates a single Huffman code with a combination of values, for example, a group of direct levels of quantized data values in a particular order. Vector Huffman encoding can lead to better bitrate reduction than scalar Huffman encoding (e.g., by allowing the encoder to exploit probabilities fractionally in binary Huffman codes). On the other hand, the codebook for vector Huffman encoding can be extremely large when single codes represent large groups of symbols or symbols have large ranges of potential values (due to the large number of potential combinations). For example, if the alphabet size is 256 (for values 0 to 255 per symbol) and the number of symbols per vector is 4, the number of potential combinations is $256^4 = 4{,}294{,}967{,}296$. This consumes memory and processing resources in computing the codebook and finding Huffman codes, and consumes transmission resources in transmitting the codebook.

Numerous variations of Huffman encoding/decoding have been developed. For additional information about Huffman encoding/decoding and some of its variations, see, e.g., Bell et al., Text Compression, Prentice Hall PTR, pages 105-107, 1990;

Gibson et al., Digital Compression for Multimedia, Morgan Kaufmann, pages 17-62, 1998.

U.S. Patent No. 6,223,162 to Chen et al. describes multi-level run-length coding of audio data. A frequency transformation produces a series of frequency coefficient values. For portions of a frequency spectrum in which the predominant value is zero, a multi-level run-length encoder statistically correlates runs of zero values with adjacent non-zero values and assigns variable length code words. An encoder uses a specialized codebook generated with respect to the probability of receiving an input run of zero-valued spectral coefficients followed by a non-zero coefficient. A corresponding decoder associates a variable length code word with a run of zero value coefficients and adjacent non-zero value coefficient.

U.S. Patent No. 6,377,930 to Chen et al. describes variable to variable length encoding of audio data. An encoder assigns a variable length code to a variable size group of frequency coefficient values.

U.S. Patent No. 6,300,888 to Chen et al. describes entropy code mode switching for frequency domain audio coding. A frequency-domain audio encoder selects among different entropy coding modes according to the characteristics of an input stream. In particular, the input stream is partitioned into frequency ranges according to statistical criteria derived from statistical analysis of typical or actual input to be encoded. Each range is assigned an entropy encoder optimized to encode that range's type of data. During encoding and decoding, a mode selector applies the correct method to the different frequency ranges. Partition boundaries can be decided in advance, allowing the decoder to implicitly know which decoding method to apply to encoded data. Or, adaptive arrangements may be used, in which boundaries are flagged in the output stream to indicate a change in encoding mode for subsequent data. For example, a partition boundary separates primarily zero quantized frequency coefficients from primarily non-zero quantized coefficients, and then applies coders optimized for such data.

For additional detail about the Chen patents, see the patents themselves.

Arithmetic coding is another well-known compression technique used for camera video and other types of content. Arithmetic coding is sometimes used in applications where the optimal number of bits to encode a given input symbol is a fractional number of bits, and in cases where a statistical correlation among certain individual input symbols exists. Arithmetic coding generally involves representing an input sequence as a single number within a given range. Typically, the number is a fractional number between 0 and 1. Symbols in the input sequence are associated with ranges occupying portions of the space between 0 and 1. The ranges are calculated based on the probability of the particular symbol occurring in the input sequence. The fractional number used to represent the input sequence is constructed with reference to the ranges. Therefore, probability distributions for input symbols are important in arithmetic coding schemes.

In context-based arithmetic coding, different probability distributions for the input symbols are associated with different contexts. The probability distribution used to encode the input sequence changes when the context changes. The context can be calculated by measuring different factors that are expected to affect the probability of a particular input symbol appearing in an input sequence. For additional information about arithmetic encoding/decoding and some of its variations, see Nelson, The Data Compression Book, "Huffman One Better: Arithmetic Coding," Chapter 5, pp. 123-65 (1992).

Various codec systems and standards use lossless compression and decompression, including versions of Microsoft Corporation's Windows Media Audio ["WMA"] encoder and decoder. Other codec systems are provided or specified by the Motion Picture Experts Group, Audio Layer 3 ["MP3"] standard, the Motion Picture Experts Group 2, Advanced Audio Coding ["AAC"] standard, and Dolby AC3. For additional information, see the respective standards or technical publications.

Whatever the advantages of prior techniques and systems for lossless compression of audio data, they do not have the advantages of the present invention.

SUMMARY

In summary, the detailed description is directed to various techniques and tools for adaptive entropy encoding and decoding of audio data. The various techniques and tools can be used in combination or independently.

In one aspect, an encoder encodes a first portion of an audio data sequence in a direct variable-dimension vector Huffman encoding mode, switches to a run-level encoding mode at a switch point, and encodes a second portion in the run-level encoding mode (e.g., context-based arithmetic encoding, Huffman coding, vector Huffman coding). For example, the first portion consists primarily of non-zero quantized audio coefficients, and the second portion consists primarily of zero-value quantized audio coefficients. The switch point can be pre-determined (e.g., by testing efficiency of encoding the sequence using the switch point) or adaptively determined. The encoder can send a flag indicating the switch point in an encoded bitstream.

In another aspect, a decoder decodes a first portion of an encoded sequence in a direct variable-dimension vector Huffman decoding mode, switches to a run-level decoding mode at a switch point, and decodes a second portion in the run-level decoding mode (e.g., context-based arithmetic decoding, Huffman decoding, vector Huffman decoding). Prior to switching, the decoder can receive a flag indicating the switch point.

In another aspect, an encoder or decoder encodes or decodes a first portion of a sequence in a direct context-based arithmetic mode, switches to a run-level mode at a switch-point, and encodes or decodes a second portion in the run-level mode. The run-level mode can be context-based arithmetic mode.

In another aspect, an encoder selects a first code table from a set of plural code tables based on the number of symbols in a first vector and represents the first vector

with a code from the first code table. The first code table can include codes for representing probable vectors having that number of symbols, and an escape code for less probable vectors. The encoder also encodes a second vector having a different number of symbols. For example, the first vector has a greater number of symbols than the second vector and has a higher probability of occurrence than the second vector. To encode the second vector, the encoder can select a second, different code table based on the number of symbols in the second vector. If the second vector has one symbol, the encoder can represent the second vector using a table-less encoding technique.

In another aspect, a decoder decodes a first vector by receiving a first code and looking up the first code in a first code table. If the first code is an escape code, the decoder receives and decodes a second code that is not in the first table. If the first code is not an escape code, the decoder looks up symbols for the first vector in the first table and includes them in a decoded data stream. The number of symbols in the first vector is the basis for whether the first code is an escape code. The decoder can decode the second code by looking it up in a second table. If the second code is an escape code, the decoder receives and decodes a third code representing the first vector that is not in the second table. If the second code is not an escape code, the decoder looks up symbols for the first vector in the second table and includes the symbols in the decoded data stream.

In another aspect, an encoder encodes audio data coefficients using a table-less encoding technique. If a coefficient is within a first value range, the encoder encodes the coefficient with a one-bit code followed by an 8-bit encoded value. For other value ranges, the encoder encodes the coefficient with a two-bit code followed by a 16-bit encoded value, a three-bit code followed by a 24-bit encoded value, or a different three-bit code followed by a 31-bit encoded value.

In another aspect, in a vector Huffman encoding scheme, an encoder determines a Huffman code from a group of such codes to use for encoding a vector and encodes the vector using the Huffman code. The determination of the code is based on a sum of

values of the audio data symbols in the vector. If the Huffman code is an escape code, it indicates that an n-dimension vector is to be encoded as x n/x-dimension vectors using at least one different code table. The encoder can compare the sum with a threshold that depends on the number of symbols in the vector. For example, the threshold is 6 for 4 symbols, 16 for 2 symbols, or 100 for 1 symbol.

In another aspect, an encoder receives a sequence of audio data and encodes at least part of the sequence using context-based arithmetic encoding. A decoder receives an encoded sequence of audio data coefficients and decodes at least part of the encoded sequence using context-based arithmetic decoding.

In another aspect, an encoder encodes audio data coefficients using context-based arithmetic coding. One or more contexts have associated probability distributions representing probabilities of coefficients. The encoder adaptively determines a context for a current coefficient based at least in part on a mode of representation of the current coefficient and encodes the current coefficient using the context. For example, if the mode of representation is direct, the encoder adaptively determines the context based at least in part on the direct levels of previous coefficients (e.g., the two coefficients immediately preceding the current coefficient). If the mode of representation is run-level, the encoder adaptively determines the context based at least in part on the percentage of zero-value coefficients the previous run length of zero-value coefficients in the audio input sequence. If the mode of representation is run-level, and the encoder adaptively determines the context based at least in part on the current run length of zero-value coefficients, the previous run length of zero-value coefficients, and the direct levels of previous coefficients.

In another aspect, an encoder or decoder encodes or decodes a first portion of audio data using direct encoding or decoding, maintaining a count of consecutive coefficients equal to a predominant value (e.g., 0). If the count exceeds a threshold, the encoder or decoder encodes or decodes a second portion of the audio data using run-level encoding or decoding. The threshold can be static or determined adaptively. The

threshold can depend on the size of the block of coefficients. For example, the threshold can be 4 for a block of 256 coefficients, or 8 for a block of 512 coefficients.

In another aspect, an encoder or decoder encodes or decodes a first portion of a sequence using a first code table and a second portion of the sequence using a second code table. The first table is used when longer runs of consecutive coefficients equal to a predominant value (e.g., 0) are more likely, and the second table is used when shorter runs of consecutive coefficients of equal value are more likely. The table that is used can be indicated by a signal bit.

The features and advantages of the adaptive entropy encoding and decoding techniques will be made apparent from the following detailed description of various embodiments that proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a suitable computing environment in which described embodiments may be implemented.

Figure 2 is a block diagram of an audio encoder in which described embodiments may be implemented.

Figure 3 is a block diagram of an audio decoder in which described embodiments may be implemented.

Figure 4 is flowchart showing a generalized multi-mode audio encoding technique.

Figure 5 is a flowchart showing a multi-mode audio encoding technique with adaptive switch point calculation.

Figure 6 is a flowchart showing a generalized multi-mode audio decoding technique.

Figure 7 is a flowchart showing a generalized variable-dimension vector Huffman encoding technique.

Figure 8 is a flowchart showing a detailed technique for encoding audio data using variable-dimension vector Huffman encoding.

Figure 9 is a flowchart showing a technique for variable-dimension vector Huffman coding of direct signal levels where the encoder adaptively determines a switch point for changing to coding of run lengths and signal levels.

Figure 10 is a flowchart showing a generalized variable-dimension vector Huffman decoding technique.

Figure 11 is a flowchart showing a detailed technique for decoding vectors coded using variable-dimension vector Huffman encoding.

Figure 12 is a flowchart showing a technique for variable-dimension vector Huffman decoding of direct signal levels where the decoder adaptively determines a switch point for changing to decoding of run lengths and signal levels.

Figures 13A-13D are probability distributions for non run-length levels in a context-based arithmetic encoding scheme.

Figures 14A-14H are probability distributions for different run lengths in a context-based arithmetic encoding scheme.

Figures 15A-15H are probability distributions for run-length encoded levels in a context-based arithmetic encoding scheme.

Figure 16 is a flowchart showing a technique for direct context-based arithmetic coding of coefficients where a switch point for changing to coding of run lengths and levels is determined adaptively by the encoder.

Figure 17 is a flowchart showing a technique for context-based arithmetic decoding where the decoder adaptively determines a switch point for changing to decoding of run lengths and signal levels.

DETAILED DESCRIPTION

In described embodiments, an audio encoder performs several adaptive entropy encoding techniques. The adaptive entropy encoding techniques improve the

performance of the encoder, reducing bitrate and/or improving quality. A decoder performs corresponding entropy decoding techniques. While the techniques are described in places herein as part of a single, integrated system, the techniques can be applied separately, potentially in combination with other techniques.

The audio encoder and decoder process discrete audio signals. In the described embodiments, the audio signals are quantized coefficients from frequency transformed audio signals. Alternatively, the encoder and decoder process another kind of discrete audio signal or discrete signal representing video or another kind of information.

In some embodiments, an audio encoder adaptively switches between coding of direct signal levels and coding of run lengths and signal levels. The encoder encodes the direct signal levels using scalar Huffman codes, vector Huffman codes, arithmetic coding, or another technique. In the run length/level coding (also called run-level coding), each run length represents a run of zero or more zeroes and each signal level represents a non-zero value. In the run-level event space, the encoder encodes run lengths and levels in that event space using Huffman codes, arithmetic coding, or another technique. A decoder performs corresponding adaptive switching during decoding. The adaptive switching occurs when a threshold number of zero value levels is reached. Alternatively, the encoder and decoder switch based upon additional or other criteria.

In some embodiments, an audio encoder uses variable-dimension vector Huffman encoding. The variable-dimension vector Huffman coding allows the encoder to use Huffman codes to represent more probable combinations of symbols using larger dimension vectors, and less probable combinations of symbols using smaller dimension vectors or scalars. A decoder performs corresponding variable-dimension Huffman decoding.

In some embodiments, an audio encoder uses context-based arithmetic coding. The contexts used by the encoder allow efficient compression of different kinds of audio data. A decoder performs corresponding context-based arithmetic decoding.

In described embodiments, the audio encoder and decoder perform various techniques. Although the operations for these techniques are typically described in a particular, sequential order for the sake of presentation, it should be understood that this manner of description encompasses minor rearrangements in the order of operations. Moreover, for the sake of simplicity, flowcharts typically do not show the various ways in which particular techniques can be used in conjunction with other techniques.

I. Computing Environment

Figure 1 illustrates a generalized example of a suitable computing environment (100) in which described embodiments may be implemented. The computing environment (100) is not intended to suggest any limitation as to scope of use or functionality of the invention, as the present invention may be implemented in diverse general-purpose or special-purpose computing environments.

With reference to Figure 1, the computing environment (100) includes at least one processing unit (110) and memory (120). In Figure 1, this most basic configuration (130) is included within a dashed line. The processing unit (110) executes computer-executable instructions and may be a real or a virtual processor. In a multi-processing system, multiple processing units execute computer-executable instructions to increase processing power. The memory (120) may be volatile memory (e.g., registers, cache, RAM), non-volatile memory (e.g., ROM, EEPROM, flash memory, etc.), or some combination of the two. The memory (120) stores software (180) implementing an audio encoder/decoder that performs adaptive entropy coding/decoding of audio data.

A computing environment may have additional features. For example, the computing environment (100) includes storage (140), one or more input devices (150), one or more output devices (160), and one or more communication connections (170). An interconnection mechanism (not shown) such as a bus, controller, or network interconnects the components of the computing environment (100). Typically, operating system software (not shown) provides an operating environment for other

software executing in the computing environment (100), and coordinates activities of the components of the computing environment (100).

The storage (140) may be removable or non-removable, and includes magnetic disks, magnetic tapes or cassettes, CD-ROMs, CD-RWs, DVDs, or any other medium which can be used to store information and which can be accessed within the computing environment (100). The storage (140) stores instructions for the software (180) implementing the audio encoder/decoder that performs adaptive entropy coding/decoding of audio data.

The input device(s) (150) may be a touch input device such as a keyboard, mouse, pen, or trackball, a voice input device, a scanning device, network adapter, or another device that provides input to the computing environment (100). For audio, the input device(s) (150) may be a sound card or similar device that accepts audio input in analog or digital form, or a CD-ROM reader that provides audio samples to the computing environment. The output device(s) (160) may be a display, printer, speaker, CD/DVD-writer, network adapter, or another device that provides output from the computing environment (100).

The communication connection(s) (170) enable communication over a communication medium to another computing entity. The communication medium conveys information such as computer-executable instructions, compressed audio information, or other data in a modulated data signal. A modulated data signal is a signal that has one or more of its characteristics set or changed to encode information in the signal. By way of example, and not limitation, communication media include wired or wireless techniques implemented with an electrical, optical, RF, infrared, acoustic, or other carrier.

The invention can be described in the general context of computer-readable media. Computer-readable media are any available media that can be accessed within a computing environment. By way of example, and not limitation, within the computing

environment (100), computer-readable media include memory (120), storage (140), communication media, and combinations of any of the above.

The invention can be described in the general context of computer-executable instructions, such as those included in program modules, being executed in a computing environment on a target real or virtual processor. Generally, program modules include routines, programs, libraries, objects, classes, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The functionality of the program modules may be combined or split between program modules as desired in various embodiments. Computer-executable instructions for program modules may be executed within a local or distributed computing environment.

For the sake of presentation, the detailed description uses terms like "analyze," "send," "compare," and "check" to describe computer operations in a computing environment. These terms are high-level abstractions for operations performed by a computer, and should not be confused with acts performed by a human being. The actual computer operations corresponding to these terms vary depending on implementation.

II. Generalized Audio Encoder and Decoder

Figure 2 is a block diagram of a generalized audio encoder (200) in which described embodiments may be implemented. The encoder (200) performs adaptive entropy coding of audio data. Figure 3 is a block diagram of a generalized audio decoder (300) in which described embodiments may be implemented. The decoder (300) decodes encoded audio data.

The relationships shown between modules within the encoder and decoder indicate a flow of information in an exemplary encoder and decoder; other relationships are not shown for the sake of simplicity. Depending on implementation and the type of compression desired, modules of the encoder or decoder can be added, omitted, split into multiple modules, combined with other modules, and/or replaced with like

modules. In alternative embodiments, encoders or decoders with different modules and/or other configurations perform adaptive entropy coding and decoding of audio data.

A. Generalized Audio Encoder

The generalized audio encoder (200) includes a selector (208), a multi-channel pre-processor (210), a partitioner/tile configurer (220), a frequency transformer (230), a perception modeler (240), a weighter (242), a multi-channel transformer (250), a quantizer (260), an entropy encoder (270), a controller (280), a mixed/pure lossless coder (272) and associated entropy encoder (274), and a bitstream multiplexer ["MUX"] (290). Description about some of the modules of the encoder (200) follows. For description about the other modules of the encoder (200) in some embodiments, see the applications referenced in the Related Application Data section.

The encoder (200) receives a time series of input audio samples (205) at some sampling depth and rate in pulse code modulated ["PCM"] format. The input audio samples (205) can be multi-channel audio (e.g., stereo mode, surround) or mono. The encoder (200) compresses the audio samples (205) and multiplexes information produced by the various modules of the encoder (200) to output a bitstream (295) in a format such as a Windows Media Audio ["WMA"] format or Advanced Streaming Format ["ASF"]. Alternatively, the encoder (200) works with other input and/or output formats.

Initially, the selector (208) selects between multiple encoding modes for the audio samples (205). In Figure 2, the selector (208) switches between two modes: a mixed/pure lossless coding mode and a lossy coding mode. The lossless coding mode includes the mixed/pure lossless coder (272) and is typically used for high quality (and high bitrate) compression. The lossy coding mode includes components such as the weighter (242) and quantizer (260) and is typically used for adjustable quality (and controlled bitrate) compression. The selection decision at the selector (208) depends

upon user input (e.g., a user selecting lossless encoding for making high quality audio copies) or other criteria. In other circumstances (e.g., when lossy compression fails to deliver adequate quality or overproduces bits), the encoder (200) may switch from lossy coding over to mixed/pure lossless coding for a frame or set of frames.

The frequency transformer (230) receives the audio samples (205) and converts them into data in the frequency domain. The frequency transformer (230) outputs blocks of frequency coefficient data to the weighter (242) and outputs side information such as block sizes to the MUX (290). The frequency transformer (230) outputs both the frequency coefficients and the side information to the perception modeler (240).

The perception modeler (240) models properties of the human auditory system to improve the perceived quality of the reconstructed audio signal for a given bitrate. Generally, the perception modeler (240) processes the audio data according to an auditory model, then provides information to the weighter (242) which can be used to generate weighting factors for the audio data. The perception modeler (240) uses any of various auditory models and passes excitation pattern information or other information to the weighter (242).

As a quantization band weighter, the weighter (242) generates weighting factors for a quantization matrix based upon the information received from the perception modeler (240) and applies the weighting factors to the data received from the frequency transformer (230). The weighter (242) outputs side information such as the set of weighting factors to the MUX (290). As a channel weighter, the weighter (242) then generates channel-specific weighting factors based on the information received from the perception modeler (240) and also on the quality of locally reconstructed signal. These scalar weights allow the reconstructed channels to have approximately uniform quality. The weighter (242) outputs weighted blocks of coefficient data to the multi-channel transformer (250) and outputs side information such as the set of channel weight factors to the MUX (290). Alternatively, the encoder (200) uses another form of weighting or skips weighting.

For multi-channel audio data, the multiple channels of noise-shaped frequency coefficient data produced by the weighter (242) often correlate. To exploit this correlation, the multi-channel transformer (250) can apply a multi-channel transform to the audio data. The multi-channel transformer (250) produces side information to the MUX (290) indicating, for example, the multi-channel transforms used and multi-channel transformed parts of frames.

The quantizer (260) quantizes the output of the multi-channel transformer (250), producing quantized coefficient data to the entropy encoder (270) and side information including quantization step sizes to the MUX (290). Quantization introduces irreversible loss of information, but also allows the encoder (200) to regulate the quality and bitrate of the output bitstream (295) in conjunction with the controller (280). In some embodiments, the quantizer (260) is an adaptive, uniform, scalar quantizer. In alternative embodiments, the quantizer is a non-uniform quantizer, a vector quantizer, and/or a non-adaptive quantizer, or uses a different form of adaptive, uniform, scalar quantization.

The entropy encoder (270) losslessly compresses quantized coefficient data received from the quantizer (260). In some embodiments, the entropy encoder (270) uses adaptive entropy encoding as described in the sections below. The entropy encoder (270) can compute the number of bits spent encoding audio information and pass this information to the rate/quality controller (280).

The controller (280) works with the quantizer (260) to regulate the bitrate and/or quality of the output of the encoder (200). The controller (280) receives information from other modules of the encoder (200) and processes the received information to determine desired quantization factors given current conditions. The controller (280) outputs the quantization factors to the quantizer (260) with the goal of satisfying quality and/or bitrate constraints.

The mixed lossless/pure lossless encoder (272) and associated entropy encoder (274) compress audio data for the mixed/pure lossless coding mode. The encoder (200)

uses the mixed/pure lossless coding mode for an entire sequence or switches between coding modes on a frame-by-frame or other basis.

The MUX (290) multiplexes the side information received from the other modules of the audio encoder (200) along with the entropy encoded data received from the entropy encoder (270). The MUX (290) outputs the information in a WMA format or another format that an audio decoder recognizes. The MUX (290) includes a virtual buffer that stores the bitstream (295) to be output by the encoder (200). The current fullness of the buffer, the rate of change of fullness of the buffer, and other characteristics of the buffer can be used by the controller (280) to regulate quality and/or bitrate for different applications (e.g., at constant quality/variable bitrate, at or below constant bitrate/variable quality).

B. Generalized Audio Decoder

With reference to Figure 3, the generalized audio decoder (300) includes a bitstream demultiplexer ["DEMUX"] (310), one or more entropy decoders (320), a mixed/pure lossless decoder (322), a tile configuration decoder (330), an inverse multi-channel transformer (340), an inverse quantizer/weighter (350), an inverse frequency transformer (360), an overlapper/adder (370), and a multi-channel post-processor (380). The decoder (300) is somewhat simpler than the encoder (300) because the decoder (300) does not include modules for rate/quality control or perception modeling. Description about some of the modules of the decoder (300) follows. For description about the other modules of the decoder (300) in some embodiments, see the applications referenced in the Related Application Data section.

The decoder (300) receives a bitstream (305) of compressed audio information in a WMA format or another format. The bitstream (305) includes entropy encoded data as well as side information from which the decoder (300) reconstructs audio samples (395).

The DEMUX (310) parses information in the bitstream (305) and sends information to the modules of the decoder (300). The DEMUX (310) includes one or more buffers to compensate for short-term variations in bitrate due to fluctuations in complexity of the audio, network jitter, and/or other factors.

The one or more entropy decoders (320) losslessly decompress entropy codes received from the DEMUX (310). For the sake of simplicity, one entropy decoder module is shown in Figure 3, although different entropy decoders may be used for lossy and lossless coding modes, or even within modes. Also, for the sake of simplicity, Figure 3 does not show mode selection logic. The entropy decoder (320) typically applies the inverse of the entropy encoding technique used in the encoder (200). When decoding data compressed in lossy coding mode, the entropy decoder (320) produces quantized frequency coefficient data.

The mixed/pure lossless decoder (322) and associated entropy decoder(s) (320) decompress losslessly encoded audio data for the mixed/pure lossless coding mode. The decoder (300) uses a particular decoding mode for an entire sequence, or switches decoding modes on a frame-by-frame or other basis.

The inverse multi-channel transformer (340) receives the entropy decoded quantized frequency coefficient data from the entropy decoder(s) (320) as well as side information from the DEMUX (310) indicating, for example, the multi-channel transform used and transformed parts of frames.

The inverse quantizer/weighter (350) receives quantization factors as well as quantization matrices from the DEMUX (310) and receives quantized frequency coefficient data from the inverse multi-channel transformer (340). The inverse quantizer/weighter (350) decompresses the received quantization factor/matrix information as necessary, then performs the inverse quantization and weighting.

The inverse frequency transformer (360) receives the frequency coefficient data output by the inverse quantizer/weighter (350) as well as side information from the DEMUX (310). The inverse frequency transformer (360) applies the inverse of the

frequency transform used in the encoder and outputs blocks to the overlapper/adder (370).

The overlapper/adder (370) receives decoded information from the inverse frequency transformer (360) and/or mixed/pure lossless decoder (322). The overlapper/adder (370) overlaps and adds audio data as necessary and interleaves frames or other sequences of audio data encoded with different modes.

III. Adaptive Entropy Encoding/Decoding Mode Switching

Run-level coding methods are often more effective than direct coding of levels when an input sequence contains many occurrences of a single value (e.g., 0). However, because non-zero quantized transform coefficients are common in audio data input sequences, especially in the lower frequencies, run-level coding is not effective across the entire range of frequencies. Moreover, in higher quality audio, non-zero quantized transform coefficients become more common even in higher frequencies. (In higher quality audio, quantization levels are typically smaller.) Therefore, in some embodiments, an encoder such as the encoder (200) of Figure 2 performs a multi-mode coding technique that can use run-level coding for one portion of an audio data input sequence and direct coding of levels for another portion of the sequence. A decoder such as the decoder (300) of Figure 3 performs a corresponding multi-mode decoding technique.

A. Adaptive Entropy Encoding Mode Switching

Referring to Figure 4, in a multi-mode encoding technique 400, the encoder first codes signal levels in an input stream directly (410). For example, the encoder performs variable-dimension Huffman coding, context-based arithmetic coding, or another entropy coding technique directly on the signal levels.

At a switch point during the encoding, the encoder changes the coding scheme (420). The encoder may change the encoding scheme at a pre-determined switch point,

or the encoder may analyze the input data to determine an appropriate point to change coding schemes. For example, the encoder may analyze an input sequence to find the best point to switch to run-level coding, sending the switch point to the decoder in the output bitstream. Or, the encoder may calculate the switch point adaptively by counting consecutive zeroes (or alternatively, another predominant value) in the input data, and switch to run-level coding when a particular threshold number of consecutive zeroes has been counted. The decoder can calculate the switch point in the same way, so the switch point need not be included in the bitstream. Or, the encoder and decoder use some other criteria to determine the switch point.

After the switch point, the encoder codes remaining signal levels using run-level coding (430). For example, the encoder performs Huffman coding, context-based arithmetic coding, or another entropy coding technique on the run lengths and signal levels. The encoder may use the same technique (e.g., context-based arithmetic coding) before and after the switch point, or the encoder may use different techniques.

Moreover, although Figure 4 and various other Figures in the application show a single switch point, additional switch points can be used to divide input data into more than two portions. For example, additional adaptive switch points can be set for increased thresholds of consecutive zeroes. Different encoding schemes can then be applied to the different portions. Or, the encoder can experiment with different segmentation points in the sequence, weighing the coding efficiencies for different segmentation configurations along with the costs of signaling the different configurations to the decoder.

Figure 5 shows a multi-mode encoding technique (500) with adaptive switch point calculation according to one implementation. The adaptive switch point depends on a count of consecutive zero-value coefficients. The input data are signal levels for quantized transform coefficients, progressing from the lowest-frequency coefficient to the highest-frequency coefficient. In practice, the position of the switch point depends

on the signal being compressed and the bitrate/quality of the encoding. Alternatively, the input data are another form and/or organization of audio data.

To start, the encoder initializes several variables. Specifically, the encoder sets a run count variable to 0 (510) and sets an encoding state variable to "direct" (512).

The encoder receives the next coefficient QC as input (520). The encoder then checks (530) if the coefficient QC is zero. If the coefficient QC is non-zero, the encoder resets the run count (538). Otherwise (i.e., if the coefficient QC is zero), the encoder increments the run count variable (532), and checks to see whether the current run count exceeds the run count threshold (534). The run count threshold can be static or it can depend on a factor such as the size of a block of coefficients (e.g., a run count threshold of 4 for a sequence of 256 coefficients, 8 for a sequence of 512 coefficients, etc.), or it can be adaptive in some other way. If the run count exceeds the threshold, the encoder changes the encoding state to run-level encoding ["RLE"] (536).

The encoder then encodes the coefficient QC if appropriate (540). (In some cases, groups of coefficients are coded together using a technique such as vector Huffman coding. In such cases, the encoder may defer encoding the coefficient QC.)

The encoder then checks (550) whether the encoder should switch encoding modes. In particular, the encoder checks the encoding state. If the encoding state is no longer direct (e.g., if the encoder has changed the encoding state to RLE as a result of reaching a threshold number of zero coefficients), the encoder begins run-level encoding of the coefficients (560). (Again, in cases in which groups of coefficients are coded together, the encoder may defer the switching decision until reaching a convenient break point for a group of coefficients.)

If the encoder does not switch encoding modes, the encoder checks whether it has finished encoding the coefficients (570). If so, the encoder exits. Otherwise, the encoder inputs the next coefficient (520) to continue the encoding process.

B. Adaptive Entropy Decoding Mode Switching

Referring to Figure 6, in a multi-mode decoding technique (600), the decoder decodes directly coded signal levels (610). For example, the decoder performs variable-dimension Huffman decoding, context-based arithmetic decoding, or another entropy decoding technique on directly coded signal levels.

At a switch point during the decoding, the decoder changes the decoding scheme (620). If the switch point is pre-determined, the decoder may receive, in the form of a flag or other notification mechanism, data that explicitly tells the decoder when to change decoding schemes. Or, the decoder may adaptively calculate when to change decoding schemes based on the input data it receives. If the decoder calculates the switch point, the decoder uses the same calculating technique used by the encoder to ensure that the decoding scheme changes at the correct point. For example, the decoder counts consecutive zeroes (or alternatively, another predominant value) to determine the switch point adaptively. In one implementation, the decoder uses a technique corresponding to the encoder technique shown in Figure 5. Or, the decoder uses some other criteria to determine the switch point.

After the switch point, the decoder decodes remaining run-level coded signal levels (630). For example, the decoder performs Huffman decoding, context-based arithmetic decoding, or another entropy decoding technique on the encoded run lengths and signal levels. The decoder may use the same technique (e.g., context-based arithmetic decoding) before and after the switch point, or the decoder may use different techniques.

IV. Variable Dimension Huffman Encoding and Decoding

While symbols such as direct signal levels can be encoded using scalar Huffman encoding, such an approach is limited where the optimal number of bits for encoding a symbol is a fractional number. Scalar Huffman coding is also limited by the inability of scalar Huffman codes to account for statistical correlation between symbols. Vector

Huffman encoding yields better bitrate reduction than scalar Huffman encoding (e.g., by allowing the encoder to exploit probabilities fractionally in binary Huffman codes). And, in general, higher-dimension vectors yield better bitrate reduction than smaller-dimension vectors. However, if a code is assigned to each possible symbol combination, codebook size increases exponentially as the vector dimension increases. For example, in a 32-bit system, the number of possible combinations for a 4-dimension vector is $(2^{32})^4$. The search time for matching a vector and finding a Huffman code also increases dramatically as codebook size increases.

In some embodiments, to reduce codebook size, an encoder such as the encoder (200) of Figure 2 uses a variable-dimension vector Huffman coding technique. Rather than assigning a codebook code to each possible n-dimensional combination, a limited number of the most probable n-dimension vectors are assigned codes. If a particular n-dimension vector is not assigned a code, the n-dimension vector is instead encoded as smaller dimension vectors (e.g., two n/2-dimension vectors), as scalars with Huffman codes, or as scalars using some table-less technique for representing discrete values. A decoder such as the decoder (300) of Figure 3 reconstructs a vector by finding the code(s) for the vector and finding the associated values.

For example, in the case of 4-dimensional vectors with 256 values possible per symbol, the encoder encodes the 500 most probable 4-dimensional vectors with Huffman codes and uses an escape code to indicate other vectors. The encoder splits such other vectors into 2-dimensional vectors. The encoder encodes the 500 most probable 2-dimensional vectors with Huffman codes and uses an escape code to indicate other vectors, which are split and coded with scalar Huffman codes. Thus, the encoder uses 501 + 501 + 256 codes.

In terms of determining which vectors or scalars are represented with Huffman codes in a table, and in terms of assigning the Huffman codes themselves for a table, codebook construction can be static, adaptive to data previously encoded, or adaptive to the data to be encoded.

A. Variable-dimension Vector Huffman Encoding

Referring to Figure 7, an encoder uses a variable-dimension vector Huffman ["VDVH"] encoding technique (700). For example, the encoder uses the technique (700) to directly encode signal levels for frequency coefficients of audio data. Alternatively, the encoder uses the technique (700) to encode another form of audio data. For the sake of simplicity, Figure 7 does not show codebook construction. Codebook construction can be static, adaptive to data previously encoded, or adaptive to the data to be encoded.

The encoder gets (710) the next vector of n symbols. For example, the encoder gets the next 4 symbols in sequence.

The encoder checks (720) whether the codebook includes a code for the vector. If so, the encoder uses (730) a single Huffman code to encode the vector. For example, to determine how to encode an n-dimension vector, the encoder checks an n-dimension vector code table for a code associated with the vector. Because larger-dimension vectors usually yield greater bitrate savings, the encoder uses Huffman codes for the most probable, n-dimension vectors. But, to limit the size of the table, only some of the n-dimension vectors have associated codes.

If the codebook does not include a code for the vector, the encoder splits (740) the vector into smaller vectors and/or scalars and codes the smaller vectors and/or scalars. For example, the encoder splits a vector of n symbols into x n/x-symbol vectors. For each n/x symbol vector, the encoder recursively repeats the encoding technique, exiting when the n/x symbol vector or its constituent vectors/scalars are encoded with Huffman codes or (for scalars) using a table-less technique for representing discrete values.

The encoder then checks (750) whether there are any additional vectors to encode. If not, the encoder exits. Otherwise, the encoder gets (710) the next vector of n symbols.

1. Example Implementation

Figure 8 shows a detailed technique (800) for encoding vectors using VDVH encoding in one implementation. In the technique (800), the encoder sums the integer values of the symbols in a vector of symbols to determine whether to encode the vector using a single Huffman code or split the vector into smaller vectors/scalars. This effectively limits codebook size and speeds up the search for codes.

A codebook table for n-dimension ["n-dim"] vectors includes Huffman codes for L_1 n-dim vectors. The codebook table also includes an escape code. The L_1 codes are for each vector for which the sum of the vector components (which are integers) is below a particular threshold T_1. For example, suppose n is 4 and the threshold T_1 for 4-dim vectors is 6. The codebook table for 4-dim vectors includes the escape code and 126 codes, one for each possible vector whose components (e.g., the absolute values of components) add up to less than 6 – (0, 0, 0, 0), (0, 0, 0, 1), etc. Limiting the table size based upon the component sum of vectors is effective because, generally, the most probable vectors are those with smaller component sums.

If the codebook table for n-dim vectors does not have a Huffman code for a particular n-dim vector, the encoder adds an escape code to the output bitstream and encodes the n-dim vector as smaller dimension vectors or scalars, looking up those smaller dimension vectors or scalars in other codebook tables. For example, the smaller dimension is n/2 unless n/2 is 1, in which case the n-dim vector is split into scalars. Alternatively, the n-dim vector is split in some other way.

The codebook table for the smaller dimension vectors includes Huffman codes for L_2 smaller dimension vectors as well as an escape code. The L_2 codes are for each vector for which the sum of the vector components is below a particular threshold T_2 for the smaller dimension table. For example, suppose the smaller dimension is 2 and the threshold T_2 for 2-dim vectors is 16. The codebook table for 2-dim vectors includes the

escape code and 136 codes, one for each possible vector whose components (e.g., the absolute values of components) add up to less than 16 – (0, 0), (0, 1), etc.

If the codebook table for smaller dimension vectors does not have a Huffman code for a particular smaller dimension vector, the encoder adds an escape code to the output bitstream and encodes the vector as even smaller dimension vectors or scalars, using other codebook tables. This process repeats down to the scalar level. For example, the split is by a power of 2 down to the scalar level. Alternatively, the vector is split in some other way.

At the scalar level, the codebook table includes Huffman codes for L_3 scalars as well as an escape code. The L_3 codes are for each scalar below a particular threshold T_3 (which assumes small values are more probable). For example, suppose the threshold T_3 for scalars is 100. The codebook table for scalars includes 100 codes and an escape code. If a scalar does not have an associated code in the scalar code table, the scalar is coded with the escape code and a value (e.g., literal) according to a table-less technique. Using all of the numerical examples given in this section, the tables would include a total of 126 + 1 + 136 + 1 + 100 + 1 = 365 codes.

The dimension sizes for tables, vector splitting factors, and thresholds for vector component sums depend on implementation. Other implementations use different vector sizes, different splitting factors, and/or different thresholds. Alternatively, an encoder uses criteria other than vector component sums to switch vector sizes/codebook tables in VDVH encoding.

With reference to Figure 8, the encoder first gets an n-dim vector (810). The n-dim vector comprises n symbols, each symbol, for example, having a value representing the quantized level for a frequency coefficient of audio data.

The encoder sums the vector components (812) and compares the sum with a threshold (820) for n-dim vectors. If the sum is less than or equal to the threshold, the encoder codes the n-dim vector with a Huffman code from a code table (822), and continues until coding is complete (824). If the sum is greater than or equal to the

threshold, the encoder sends an escape code (826) and splits the n-dim vector into two smaller vectors with dimensions of n/2 (830).

The encoder gets the next n/2-dim vector (840) and sums the components of the n/2-dim vector (842). The encoder checks the sum against a threshold associated with n/2-dim vectors (850). If the sum is less than or equal to the threshold, the encoder codes the n/2-dim vector with a Huffman code from a code table (852) for n/2-dim vectors, and gets the next n/2-dim vector (840) if the encoder has not finished encoding the n/2-dim vectors (854). If the sum is greater than the threshold for n/2-dim vectors, the encoder sends another escape code (856).

The encoder generally follows this pattern in processing the vectors, either coding each vector or splitting the vector into smaller-dimension vectors. In cases where the encoder splits a vector into two scalar (1-dimension) components (860), the encoder gets the next scalar (870) and compares the value of the scalar with a threshold associated with scalar values (880). If the scalar value is less than or equal to the threshold (880), the encoder codes the scalar using a Huffman code from a code table (882) for scalars. If the scalar value is greater than the threshold, the encoder codes the scalar using a table-less technique (884). The encoder then gets the next scalar (870) if it has not finished processing the scalars (886).

Alternatively, the encoder uses tables with different dimension sizes, splits vectors in some way other than by power of 2, and/or uses a criteria other than vector component sum to switch vector sizes/codebook tables in VDVH encoding.

2. Adaptive Switching

Figure 9 shows a technique (900) for VDVH coding of coefficients of direct signal levels where the encoder adaptively determines a switch point for changing to coding of run lengths and signal levels according to one implementation. The adaptive switch point depends on a count of consecutive zero-value coefficients. The input data are signal levels for quantized transform coefficients, progressing from the lowest-

frequency coefficient to the highest-frequency coefficient. Alternatively, the input data are another form and/or organization of audio data.

To start, the encoder initializes several variables. Specifically, the encoder sets a run count variable to 0 (910), sets a current vector variable to empty (912), and sets an encoding state variable to direct variable-dimension vector Huffman ["DVDVH"] (914).

The encoder receives the next coefficient QC as input (920). The encoder then checks (930) if the coefficient is zero. If the coefficient QC is non-zero, the encoder resets the run count (938) and adds the coefficient QC to the current vector (940). Otherwise (i.e., if the coefficient QC is zero), the encoder increments the run count variable (932), and checks to see whether the current run count exceeds the run count threshold (934). The run count threshold can be static or it can depend on a factor such as the size of a block of coefficients (e.g., four zeroes in an input sequence of 256 coefficients), or it can be adaptive in some other way. For example, the threshold may be increased or decreased, with or without regard to the number of coefficients in an input sequence. If the run count exceeds the threshold, the encoder changes the encoding state to run-level encoding ["RLE"] (936), and the coefficient QC is added as a component to the current vector (940).

Adding the coefficient QC to the current vector increments the dimension of the vector. The encoder determines (950) whether the current vector is ready to encode by comparing the number of components in the current vector with the maximum dimension for the current vector. If so, the encoder encodes the current vector using DVDVH coding (960). If the current vector is smaller than the maximum dimension, but the coefficient QC is the last in a sequence, the encoder can pad the current vector and encode it using DVDVH coding (960). The maximum dimension depends on implementation. In one implementation, it is 8. However, the maximum dimension may be increased or decreased depending on, for example, the amount of resources available for creating, storing or transmitting a codebook.

After encoding the vector, the encoder checks the encoding state (970). If the encoding state is no longer DVDVH (e.g., if the encoder has changed the encoding state to RLE as a result of exceeding a threshold number of zero coefficients), the encoder begins encoding of the coefficients as run lengths and levels (980). Run-level encoding can be performed in several ways, including, for example, Huffman coding, vector Huffman coding, or context-based arithmetic coding. In some embodiments, run-level encoding is performed using Huffman coding with two Huffman code tables, where one table is used for encoding data in which shorter runs are more likely, and one table is used for encoding data in which longer runs are more likely. The encoder tries each table, and chooses codes from one of the tables, with a signal bit indicating which table the encoder used.

If the encoding state has not changed or the current vector is not ready for encoding, the encoder determines (990) whether there are any more coefficients to be encoded. If so, the encoder inputs the next coefficient (920) and continues the encoding process.

B. Variable-dimension Vector Huffman Decoding

Figure 10 shows a VDVH decoding technique (1000) corresponding to the VDVH encoding technique (700) shown in Figure 7. For example, a decoder uses the technique (1000) to decode directly encoded signal levels for frequency coefficients of audio data. Alternatively, the decoder uses the technique to decode another form of audio data.

The decoder gets (1010) the next Huffman code for an n-dimension vector Huffman coding table. For example, the decoder gets the next Huffman code for 4 symbols in sequence.

The decoder checks (1020) whether the Huffman code is the escape code for the n-dimension vector Huffman coding table. If not, the decoder gets (1030) the n symbols represented by the Huffman code. For example, the decoder gets the 4

symbols associated with the Huffman code in a 4-dimensional vector Huffman codebook.

If code is the escape code, the n-dimension codebook does not include a code for the vector, and the decoder gets (1040) Huffman codes for smaller vectors and/or scalars. For example, the decoder gets codes for x n/x-symbol vectors. For each n/x symbol vector, the decoder recursively repeats the decoding technique, exiting when the n/x symbol vector or its constituent vectors/scalars are decoded.

The decoder then checks (1050) whether there are any additional codes for the n-dimension vector Huffman coding table to decode. If not, the decoder exits. Otherwise, the decoder gets (1010) the next such Huffman code.

1. Example Implementation

Figure 11 shows a detailed technique (1100) for decoding vectors coded using VDVH encoding in one implementation. The decoding technique (1100) corresponds to the encoding technique (800) shown in Figure 8.

Referring to Figure 11, the decoder gets the next code for an n-dim vector Huffman code table (1110). The decoder checks if the code is the escape code for the n-dim vector Huffman code table (1120). If not, the decoder gets the n symbols represented by the code in the n-dim vector table (1122). The decoder continues until the decoder has finished processing the encoded data (1124).

If the code is the escape code for the n-dim vector Huffman code table, the decoder decodes the n-dim vector as two n/2-dim vectors using a n/2-dim vector Huffman code table. Specifically, the decoder gets the next code for the n/2-dim vector Huffman code table (1130). The decoder checks if the code is the escape code for the n/2-dim vector Huffman code table (1140). If not, the decoder gets the n/2 symbols represented by the code in the n/2-dim vector Huffman code table (1142). The decoder continues processing the codes for the n/2-dim vector Huffman code table until the processing of such codes is complete (1144).

If the code is the escape code for the n/2-dim vector Huffman code table, the decoder decodes the n/2-dim vector as two n/4-dim vectors, which may be scalars, etc.

The decoder generally follows this pattern of decoding larger-dimension vectors as two smaller-dimension vectors when escape codes are detected, until the vectors to be decoded are scalars (1-dim vectors). At that point, the decoder gets the next code for a scalar Huffman code table (1150). The decoder checks if the code is the escape code for the scalar Huffman code table (1160). If not, the decoder gets the scalar represented by the code in the scalar Huffman code table (1162). The decoder continues processing the codes for the scalars until processing of such codes is complete (1164). If the code is the escape code for the scalar Huffman code table, the scalar is coded using a table-less technique, and the decoder gets the value (1170).

Alternatively, the decoder uses tables with different dimension sizes and/or uses tables that split vectors in some way other than by power of 2 in VDVH decoding.

2. Adaptive Switching

Figure 12 shows a technique (1200) for decoding vectors encoded using VDVH encoding according to one implementation, where the decoder adaptively determines a switch point for changing to decoding of run lengths and signal levels. The adaptive switch point depends on a count of consecutive zero-value coefficients in the data, which are signal levels for quantized transform coefficients, progressing from the lowest-frequency coefficient to the highest-frequency coefficient. Alternatively, the data are another form and/or organization of audio data.

To start, the decoder initializes several variables. Specifically, the decoder sets a run count to 0 (1210) and sets a decoding state to DVDVH (1212).

The decoder decodes the next vector by looking up the code for that vector in a Huffman coding table (1220). For example, the decoder performs the decoding technique (1100) shown in Figure 11. The decoder then updates the run count based on

the decoded vector (1230) (specifically, using the number of zero values in the decoded vector to reset, increment, or otherwise adjust the run count).

The decoder checks if the run count exceeds a threshold (1240). The run count threshold can be static or it can depend on a factor such as the size of a block of coefficients (e.g., four zeroes in an input sequence of 256 coefficients), or it can be adaptive in some other way. If the run count exceeds the threshold, the decoder begins decoding the encoded coefficients using run-level decoding (1250). Run-level decoding can be performed in several ways, including, for example, Huffman decoding, vector Huffman decoding, or context-based arithmetic decoding.

In some embodiments, run-level decoding is performed using Huffman decoding with two potential Huffman code tables, where one table is used for decoding data in which shorter runs are more likely, and one table is used for decoding data in which longer runs are more likely. When the decoder receives a code, a signal bit in the code indicates which table the encoder used, and the decoder looks up the code in the appropriate table.

If the run count does not exceed the threshold, the decoder continues processing vectors until decoding is finished (1260).

V. Context-based Arithmetic Coding and Decoding

In some embodiments, an encoder such as the encoder (200) of Figure 2 uses context-based arithmetic ["CBA"] coding to code sequences of audio data. In CBA coding, different probability distributions for the input symbols are associated with different contexts. The probability distribution used to encode the input sequence changes when the context changes. The context can be calculated by measuring different factors that are expected to affect the probability of a particular input symbol appearing in an input sequence. A decoder such as the decoder (300) of Figure 3 performs corresponding arithmetic decoding.

When encoding coefficients directly (i.e., as direct levels), the encoder uses factors including the values of the previous coefficients in the sequence to calculate the context. When encoding coefficients using run-level encoding, the encoder uses factors including the lengths of the current run and previous runs, in addition to the values of previous coefficients, to calculate the context. The encoder uses a probability distribution associated with the calculated context to determine the appropriate arithmetic code for the data. Thus, by using the various factors in calculating contexts, the encoder determines contexts adaptively with respect to the data and with respect to the mode (i.e., direct, run-level) of representation of the data.

In alternative embodiments, the encoder may use additional factors, may omit some factors, or may use the factors mentioned above in other combinations.

A. Example Implementation of Contexts

Tables 2-5 and Figures 13A – 13D , 14A – 14H, and 15A – 15H show contexts and probability distributions, respectively, used in CBA encoding and decoding in an example implementation. Alternatively, CBA encoding and decoding use different contexts and/or different probability distributions.

Although the following discussion focuses on context calculation in the encoder in the example implementation, the decoder performs corresponding context calculation during decoding using previously decoded audio data.

As noted above, the encoder can encode coefficients using CBA encoding whether the encoder is coding direct levels only or run lengths and direct levels. In one implementation, however, the techniques for calculating contexts vary depending upon whether the encoder is coding direct levels only or run lengths and direct levels. In addition, when coding run lengths and direct levels, the encoder uses different contexts depending on whether the encoder is encoding a run length or a direct level.

The encoder uses a four-context system for calculating contexts during arithmetic encoding of direct levels using causal context. The encoder calculates the

context for a current level L[n] based on the value of the previous level (L[n-1]) and the level just before the previous level (L[n-2]). This context calculation is based on the assumptions that 1) if previous levels are low, the current level is likely to be low, and 2) the two previous levels are likely to be better predictors of the current level than other levels. Table 2 shows the contexts associated with the values of the two previous levels in the four-context system. Figures 13A-13D show probability distributions for current levels for these contexts.

L[n-1]	L[n-2]	Context
= 0	= 0	0
= 0	≥1	1
= 1	Any	2
≥2	Any	3

Table 2: Contexts for CBA encoding/decoding of direct levels

The probability distributions in Figures 13A-13D assume that when the two previous levels are zero or near-zero, the current level is more likely to be zero or near-zero.

The encoder also can use CBA coding when performing run-length coding of levels. When encoding a run length, factors used by the encoder to calculate context include the percentage of zeroes in the input sequence (a running total over part or all of the sequence) and the length of the previous run of zeroes (R[n-1]). The encoder calculates a zero percentage index based on the percentage of zeroes in the input sequence, as shown below in Table 3:

Zero %	Zero % index
≥90	0
≥80	1
≥60	2
< 60	3

Table 3: Zero percentage indices for CBA encoding/decoding of run lengths

The encoder uses the zero percentage index along with the length of the previous run to calculate the context for encoding the current run length, as shown

below in Table 4. Figures 14A-14H show probability distributions for different run-length values associated with these contexts.

Zero % index	R[n-1]	Context
0	= 0	0
0	> 0	4
1	= 0	1
1	> 0	5
2	= 0	2
2	> 0	6
3	= 0	3
3	> 0	7

Table 4: Contexts for CBA encoding/decoding of run lengths

For example, in an input sequence where 91% of the levels are zeroes (resulting in a zero percentage index of 0), and where the length of the previous run of zeroes was 15, the context is 4. The probability distributions in Figures 14A-14H show that when the percentage of zeroes in an input sequence is higher, longer run lengths are more likely. The probability distributions also assume that within a given zero percentage index, run lengths following a run length of zero are likely to be shorter than run lengths following a run length greater than zero.

When encoding a level in run-level data, factors used by the encoder to calculate context include the length of the current run (R[n]), the length of the previous run (R[n-1]), and the values of the two previous levels (L[n-1] and L([n-2]). This context calculation is based on the observation that the current level is dependent on the previous two levels as long as the spacing (i.e., run lengths) between the levels is not too large. Also, if previous levels are lower, and if previous runs are shorter, the current level is likely to be low. When previous runs are longer, the previous level has less effect on the current level.

The contexts associated with the values of the current run length, previous run length, and the two previous levels are shown below in Table 5. Figures 15A-15H show probability distributions for levels associated with these contexts.

R[n]	R[n-1]	L[n-1]	L[n-2]	Context
≥2	Any	Any	Any	0
< 2	≥2	= 1	Any	1
< 2	≥2	= 2	Any	2
< 2	≥2	> 2	Any	3
< 2	< 2	= 1	= 1	4
< 2	< 2	= 1	> 1	5
< 2	< 2	= 2	Any	6
< 2	< 2	> 2	Any	7

Table 5: Contexts for CBA encoding/decoding of levels in run-level encoding

For example, in an input sequence where the length of the current run of zeroes is 1, the length of the previous run of zeroes is 2, and the previous level is 1, the context is 1. The probability distributions in Figures 15A-15H show that when the previous levels are lower, and when current and previous run lengths are shorter, the current level is more likely to be zero or near zero.

B. Adaptive Switching

Figure 16 shows a technique (1600) for CBA coding of coefficients of direct signal levels where the encoder adaptively determines a switch point for changing to coding of run lengths and signal levels according to one implementation. The adaptive switch point depends on a count of consecutive zero-value coefficients. The input data are signal levels for quantized transform coefficients, progressing from the lowest-frequency coefficient to the highest-frequency coefficient. Alternatively, the input data are another form and/or organization of audio data.

To start, the encoder initializes several variables. Specifically, the encoder sets a run count variable to 0 (1610) and sets an encoding state variable to direct context-based arithmetic (DCBA) (1612).

The encoder receives the next coefficient QC as input (1620). The encoder then checks (1630) if the coefficient is zero. If the coefficient QC is non-zero, the encoder resets the run count (1638) and codes the coefficient using DCBA encoding (1640).

Otherwise (i.e., if the coefficient QC is zero), the encoder increments the run count variable (1632), and checks to see whether the current run count exceeds the run count threshold (1634). The run count threshold can be static or it can depend on a factor such as the size of a block of coefficients (e.g., four zeroes in an input sequence of 256 coefficients), or it can be adaptive in some other way. For example, the threshold may be increased or decreased, with or without regard to the number of coefficients in an input sequence. If the run count exceeds the threshold, the encoder changes the encoding state to run-level encoding ["RLE"] (1636). The encoder then codes the coefficient using DCBA encoding (1640).

After encoding the coefficient, the encoder checks the encoding state (1650). If the encoding state is no longer DCBA (e.g., if the encoder has changed the encoding state to RLE as a result of exceeding a threshold number of zero coefficients), the encoder begins encoding of the coefficients as run lengths and levels (1660). Run-level encoding can be performed in several ways, including, for example, Huffman coding, vector Huffman coding, or CBA coding (potentially with different contexts than the earlier CBA coding, as described above). In some embodiments, run-level encoding is performed using Huffman coding with two Huffman code tables, where one table is used for encoding data in which shorter runs are more likely, and one table is used for encoding data in which longer runs are more likely. The encoder tries each table, and chooses codes from one of the tables, with a signal bit indicating which table the encoder used.

If the encoding state has not changed, the encoder determines (1670) whether there are any more coefficients to be encoded. If so, the encoder inputs the next coefficient (1620) and continues the encoding process.

C. Context-based Arithmetic Decoding

Figure 17 shows a technique (1700) for decoding coefficients encoded using CBA encoding according to one implementation, where the decoder adaptively determines a switch point for changing to decoding of run lengths and signal levels. The adaptive switch point depends on a count of consecutive zero-value coefficients in the data, which are signal levels for quantized transform coefficients, progressing from the lowest-frequency coefficient to the highest-frequency coefficient. Alternatively, the data are another form and/or organization of audio data.

To start, the decoder initializes several variables. Specifically, the decoder sets a run count to 0 (1710) and sets a decoding state to direct context-based arithmetic (DCBA) (1712).

The decoder decodes the next quantized coefficient using DCBA (1720) by looking at the number the encoder used to represent the coefficient in arithmetic encoding, and extracting the value of the coefficient from that number. The decoder then updates the run count based on the decoded coefficient (1730) (specifically, based on whether the decoded coefficient is a zero value to reset or increment the run count).

The decoder checks if the run count exceeds a threshold (1740). The run count threshold can be static or it can depend on a factor such as the size of a block of coefficients (e.g., four zeroes in an input sequence of 256 coefficients), or it can be adaptive in some other way. If the run count exceeds the threshold, the decoder begins decoding the encoded coefficients using run-level decoding (1750). Run-level decoding can be performed in several ways, including, for example, Huffman decoding, vector Huffman decoding, or CBA decoding (potentially with different contexts than the earlier CBA decoding, as described above). In some embodiments, run-level decoding is performed using Huffman decoding with two potential Huffman code tables, where one table is used for decoding data in which shorter runs are more likely, and one table is used for decoding data in which longer runs are more likely. When the decoder

receives a code, a signal bit in the code indicates which table the encoder used, and the decoder looks up the code in the appropriate table.

If the run count does not exceed the threshold, the decoder continues processing coefficients until decoding is finished (1760).

VI. Table-less Coding

In some embodiments using Huffman coding, an encoder such as the encoder (200) of Figure 2 uses an escape code for a Huffman code table to indicate that a particular symbol (or combination of symbols) does not have an associated code in the table. Sometimes, an escape code is used to indicate that a particular symbol (e.g., a scalar value for a level that is not represented in a scalar Huffman code table for levels, a run length that is not represented in a scalar Huffman code table for run lengths, etc.) is to be encoded without using a code from a Huffman table. In other words, the symbol is to be encoded using a "table-less" coding technique.

In some embodiments using arithmetic coding, an escape code is sometimes used to indicate that a particular symbol is not to be coded arithmetically. The symbol could be encoded using a code from a Huffman table, or it could also be encoded using a "table-less" encoding technique.

Some table-less coding techniques use fixed-length codes to represent symbols. However, using fixed-length codes can lead to unnecessarily long codes.

In some embodiments, therefore, symbols such as quantized transform coefficients are represented with variable length codes in a table-less encoding technique when the symbols are not otherwise encoded. A decoder such as the decoder (300) of Figure 3 performs a corresponding table-less decoding technique.

For example, Table 6 shows pseudo-code for one implementation of such a table-less encoding technique.

```
If (value < 2^8) {
        Send "0";
        Send value using 8 bits;
}
        else if (value < 2^16) {
                Send "10";
                Send value using 16 bits
        }
                else if (value < 2^24) {
                        Send "110";
                        Send value using 24 bits;
                }
                        else if (value < 2^31) {
                                Send "111";
                                Send value using 31 bits;
                        }
```

Table 6: Pseudo-code for table-less coding technique in one implementation

The number of bits the encoder uses to encode the coefficient depends on the value of the coefficient. The encoder sends a one, two, or three-bit value to indicate the number of bits used to encode the value, and then sends the encoded value itself using 8, 16, 24 or 31 bits. The total number of bits the encoder uses to encode the coefficient ranges from 9 bits for a value less than 2^8 to 34 bits for a value greater than or equal to 2^{24}, but less than 2^{31}.

For a series of coefficients, the average bits sent will be equal to:

$$P(0 \leq C < 2^8) * 9 + P(2^8 \leq C < 2^{16}) * 18 + P(2^{16} \leq C < 2^{24}) * 27 + P(2^{24} \leq C < 2^{31}) * 34,$$

where $P(m \leq C < n)$ is the probability of occurrence in an input sequence of a coefficient C within the range indicated. Significant bit savings are therefore possible when a large percentage of coefficients are small (e.g., less than 2^{16}).

Alternatively, the encoder and decoder use another table-less encoding/decoding technique.

Having described and illustrated the principles of our invention with reference to various described embodiments, it will be recognized that the described embodiments can be modified in arrangement and detail without departing from such principles. It should be understood that the programs, processes, or methods described herein are not related or limited to any particular type of computing environment, unless indicated otherwise. Various types of general purpose or specialized computing environments may be used with or perform operations in accordance with the teachings described herein. Elements of the described embodiments shown in software may be implemented in hardware and vice versa.

In view of the many possible embodiments to which the principles of our invention may be applied, we claim as our invention all such embodiments as may come within the scope and spirit of the following claims and equivalents thereto.